链滴

Shiro 使用多个 Realm 实现多种登录方式

作者：wuuJiawei
原文链接：https://ld246.com/article/1582123725664
来源网站：链滴

前言

大部分场景下，我们都会在项目中实现自定义 Realm 搭配 UsernamePasswordToken 来完成用户登录认证流程，但是如果登录方式包括“第三方登录”、“手机号登录”等，仅凭 UsernamePasswodToken 就难以实现了，因为以上的两种登录方式都是免密登录，而 UsernamePasswordToken 却须要有username和password，因此需要自定义多个Realm和Token才能实现上述功能。

本文只会实现第三方登录，以此为例，列位看官可以尝试修改代码实现自己的业务逻辑。

另外，本文篇幅较长，代码量较多，看官们一定要耐心看完，万一漏写或写错了部分代码，耽误的可更多的时间。

创建自定义Token

网上很多文章都继承 UsernamePasswordToken 来创建自己的Token，但我不建议这样写，如果继承sernamePasswordToken ，在后面的操作中会变得相对麻烦。

我们直接查看 UsernamePasswordToken 的源码，可以看到它实现了 HostAuthenticationToken 和ememberMeAuthenticationToken ，而这两个类又分别实现了 AuthenticationToken ，因此在这我们直接实现 AuthenticationToken 即可，同时重写 getPrincipal() 和 getCredentials() 两个方法。

```
import org.apache.shiro.authc.AuthenticationToken;

/**
 * 第三方授权登录凭证
 * 注意这里要实现AuthenticationToken，不能继承UsernamePasswordToken
 * 同时重写getPrincipal()和getCredentials()两个方法
 * @author wujiawei0926@yeah.net
 */
public class OAuth2UserToken implements AuthenticationToken {

    /**
     *  授权类型
     *  这里可以使用枚举
      */
    private String type;

    // 第三方登录后获取的用户信息
    private OAuth2User user;

    public OAuth2UserToken(final String type, final OAuth2User user) {
        this.type = type;
        this.user = user;
    }

    @Override
    public Object getPrincipal() {
        return this.getUser();
    }

    @Override
    public Object getCredentials() {
        return this.getUser().getOpenid();
```

```
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public OAuth2User getUser() {
    return user;
}

public void setUser(OAuth2User user) {
    this.user = user;
}

/**
 * 用户信息类，用于新用户注册
 * 可根据自己的具体业务进行拓展
 */
public static class OAuth2User {

    public OAuth2User(){};

    private String openid;
    private String username;
    private String nickname;
    private String avatar;
    private String email;
    private String remark;
    private Integer sex;

    public String getOpenid() {
        return openid;
    }

    public void setOpenid(String openid) {
        this.openid = openid;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getNickname() {
        return nickname;
    }
```

```
    public void setNickname(String nickname) {
        this.nickname = nickname;
    }

    public String getAvatar() {
        return avatar;
    }

    public void setAvatar(String avatar) {
        this.avatar = avatar;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getRemark() {
        return remark;
    }

    public void setRemark(String remark) {
        this.remark = remark;
    }

    public Integer getSex() {
        return sex;
    }

    public void setSex(Integer sex) {
        this.sex = sex;
    }
}


}
```

创建多个Realm

创建Realm时，必须重写 supports() 方法，在后面起到了至关重要的作用。

作为演示，本文创建的Realm都没有做权限的授权，即 doGetAuthorizationInfo() 没有做具体的实，列位看官需要加上自己的权限授权业务。

首先是传统的 UserRealm ，相信列位看官对这个类都很熟悉，因此这里不再赘述，直接贴上代码：

```
import com.*.dao.UserDao;
import com.*.model.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
```

```
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;

/**
 * Shiro认证和授权
 * @author wujiawei0926@yeah.net
 */
public class UserRealm extends AuthorizingRealm {

    @Autowired
    private UserDao userService;

    /**
     * 一定要重写support()方法，在后面的身份验证器中会用到
     * @param token
     * @return
     */
    @Override
    public boolean supports(AuthenticationToken token) {
        return token instanceof UsernamePasswordToken;
    }

    /**
     * 授权
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection)
{
        User user = (User) SecurityUtils.getSubject().getPrincipal();
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        return authorizationInfo;
    }

    /**
     * 认证
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticatio
Token) throws AuthenticationException {
        String username = (String) authenticationToken.getPrincipal();
        User user = userService.findByUsername(username);
        if (user == null) {
            throw new UnknownAccountException(); // 账号不存在
        }
        if (user.getDisabled() == 1) {
            throw new LockedAccountException();
        }
        if (user.getUserType() != 0) {
            throw new ConcurrentAccessException();
        }
        SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(user, user.
```

```
etPassword(), getName());
        return authenticationInfo;
    }
}
```

然后创建我们的免密登陆Realm，与 UserRealm 相同，继承 AuthorizingRealm即可，同样的，也重写 supports() 方法，并且再多重写一个 getName()方法，在后面也会用到。

用户登录的方法写在 doGetAuthenticationInfo 中，通过校验 openid，实现老用户的登录和新用户注册，下面贴上代码：

```
import com.*.dao.UserDao;
import com.w.module.base.model.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
* 自定义第三方登录授权Realm
* @author wujiawei0926@yeah.net
*/
@Component
public class OAuth2UserRealm extends AuthorizingRealm {

    public static final String REALM_NAME = "oauth2_user_realm";

    @Autowired
    private UserDao userDao;

    @Override
    public String getName() {
        return REALM_NAME;
    }

    /**
     * 检查是否支持该Realm
     * 一定要重写support()方法，在后面的身份验证器中会用到
     * @param token
     * @return
     */
    @Override
    public boolean supports(AuthenticationToken token) {
        return token instanceof OAuth2UserToken;
    }

    /**
     * 授权
     * @param principalCollection
```

```
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection)

        User user = (User) SecurityUtils.getSubject().getPrincipal();
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        return authorizationInfo;
    }

    /**
     * 认证
     * 在这个方法中，完成老用户的登录与新用户的注册
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authentication
oken) throws AuthenticationException {
        OAuth2UserToken token = (OAuth2UserToken)authenticationToken;
        OAuth2UserToken.OAuth2User oAuth2User = token.getUser();

        // 校验openid
        if (oAuth2User == null) {
            throw new AuthenticationException();
        }

        // 根据openid查询用户数据
        String openid = oAuth2User.getOpenid();
        User user = null;
        switch (token.getType()) {
            case "qq":
                user = userDao.findByQqOpenid(openid);
                break;
            case "weixin":
                user = userDao.findByWxOpenid(openid);
                break;
            default:
                break;
        }

        if (user == null) {
            // TODO 获取oAuth2User中用户信息进行注册
        }
        if (user.getDisabled() == 1) {
            // 账号被拉黑
            throw new LockedAccountException();
        }

        // 完成登录，注意这里传的principal和credentials
        // principal: OAuth2UserToken类中getPrincipal()的返回值
        // credentials: OAuth2UserToken类中getCredentials()的返回值
        SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(user, open
```

```
d, getName());
        return authenticationInfo;
    }
}
```

创建自定义ModularRealmAuthenticator

Token 和 Realm创建完成之后，需要再创建一个 ModularRealmAuthenticator 来进行绑定操作，程序知道碰到这个 Token 时进入对应的 Realm。

这步其实很简单，只需要重写 doMultiRealmAuthentication(Collection<Realm> realms, AuthenticationToken token) 方法，该方法有两个参数，其中realms是ShiroConfig中配置的所有realm集合，oken 就是登录时传入的用户信息token，在我们这边只会是UsernamePasswordToken或OAuth2UsrToken。

在doMultiRealmAuthentication()方法中遍历所有的realm，通过每个realm的 supports() 方法来进匹配。

```
package com.w.common.config.shiro;

import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.pam.ModularRealmAuthenticator;
import org.apache.shiro.authc.pam.UnsupportedTokenException;
import org.apache.shiro.realm.Realm;

import java.util.Collection;

/**
 * 自定义身份验证器，根据登录使用的Token匹配调用对应的Realm
 * @author wujiawei0926@yeah.net
 */
public class CustomModularRealmAuthenticator extends ModularRealmAuthenticator {

    /**
     * 自定义Realm的分配策略
     * 通过realm.supports()方法匹配对应的Realm，因此才要在Realm中重写supports()方法
     * @param realms
     * @param token
     * @return
     */
    @Override
    protected AuthenticationInfo doMultiRealmAuthentication(Collection<Realm> realms, Aut
enticationToken token) {
        // 判断getRealms()是否返回为空
        assertRealmsConfigured();

        // 通过supports()方法，匹配对应的Realm
        Realm uniqueRealm = null;
        for (Realm realm : realms) {
            if (realm.supports(token)) {
                uniqueRealm = realm;
                break;
```

```
            }
        }
        if (uniqueRealm == null) {
            throw new UnsupportedTokenException();
        }
        return uniqueRealm.getAuthenticationInfo(token);
    }
}
```

配置ShiroConfig

ShiroConfig中需要添加或修改一下配置：

1. modularRealmAuthenticator，告诉Shiro，以后使用我们自定义的身份验证器：

```
    /**
     * 针对多Realm，使用自定义身份验证器
     * @return
     */
    @Bean
    public ModularRealmAuthenticator modularRealmAuthenticator(){
        CustomModularRealmAuthenticator authenticator = new CustomModularRealmAuthenti
ator();
        authenticator.setAuthenticationStrategy(new AtLeastOneSuccessfulStrategy());
        return authenticator;
    }
```

2. realm，把创建的所有realm都注册到bean中：

```
    /**
     * 免密授权登录
     * @return
     */
    @Bean
    public OAuth2UserRealm oAuth2UserRealm(){
        OAuth2UserRealm realm = new OAuth2UserRealm();
        // 不需要加密，直接返回
        return realm;
    }

    @Bean
    public UserRealm userRealm() {
        UserRealm userRealm = new UserRealm();
        userRealm.setCredentialsMatcher(credentialsMatcher());
        return userRealm;
    }
```

3. securityManager，让安全管理器使用我们创建的身份验证器，并添加所有的realm：

```
    @Bean(name = "securityManager")
    public DefaultWebSecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        securityManager.setCacheManager(cacheManager());
```

```
        securityManager.setSessionManager(sessionManager());

        // 设置验证器为自定义验证器
        securityManager.setAuthenticator(modularRealmAuthenticator());
        // 设置Realms
        List<Realm> realms = new ArrayList<>(2);
        realms.add(userRealm());
        realms.add(oAuth2UserRealm());
        securityManager.setRealms(realms);

        return securityManager;
    }
```

最后把我的ShiroConfig完整代码奉上，各位参考即可：

```
import com.*.config.filter.MyLoginFilter;
import com.*.config.filter.MyLogoutFilter;
import org.apache.shiro.authc.pam.AtLeastOneSuccessfulStrategy;
import org.apache.shiro.authc.pam.ModularRealmAuthenticator;
import org.apache.shiro.cache.ehcache.EhCacheManager;
import org.apache.shiro.realm.Realm;
import org.apache.shiro.session.mgt.eis.MemorySessionDAO;
import org.apache.shiro.session.mgt.eis.SessionDAO;
import org.apache.shiro.spring.LifecycleBeanPostProcessor;
import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;
import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.DependsOn;
import org.springframework.web.filter.DelegatingFilterProxy;

import javax.servlet.Filter;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

/**
 * shiro框架配置
 */
@Configuration
public class ShiroConfig {

    @Bean(name = "shiroFilter")
    public ShiroFilterFactoryBean shiroFilter(DefaultWebSecurityManager securityManager) {
        ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();
        shiroFilter.setSecurityManager(securityManager);
        // 登录配置
        shiroFilter.setLoginUrl("login");
        shiroFilter.setSuccessUrl("system");
```

```
        shiroFilter.setUnauthorizedUrl("/error?code=403");
        // 自定义过滤器
        Map<String, Filter> filtersMap = new LinkedHashMap<>();
        filtersMap.put("access", new MyLoginFilter());
        filtersMap.put("mylogout", new MyLogoutFilter());
        shiroFilter.setFilters(filtersMap);
        // 拦截配置
        Map<String, String> filterChainDefinitions = new LinkedHashMap<>();
        filterChainDefinitions.put("/logout", "mylogout");
        filterChainDefinitions.put("/system/**", "access,authc");
        filterChainDefinitions.put("/**", "anon");
        shiroFilter.setFilterChainDefinitionMap(filterChainDefinitions);
        return shiroFilter;
    }

    @Bean(name = "securityManager")
    public DefaultWebSecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        securityManager.setCacheManager(cacheManager());
        securityManager.setSessionManager(sessionManager());

        // 设置验证器为自定义验证器
        securityManager.setAuthenticator(modularRealmAuthenticator());
        // 设置Realms
        List<Realm> realms = new ArrayList<>(2);
        realms.add(userRealm());
        realms.add(oAuth2UserRealm());
        securityManager.setRealms(realms);

        return securityManager;
    }

    /**
     * 针对多Realm，使用自定义身份验证器
     * @return
     */
    @Bean
    public ModularRealmAuthenticator modularRealmAuthenticator(){
        CustomModularRealmAuthenticator authenticator = new CustomModularRealmAuthenti
ator();
        authenticator.setAuthenticationStrategy(new AtLeastOneSuccessfulStrategy());
        return authenticator;
    }

    /**
     * 免密授权登录
     * @return
     */
    @Bean
    public OAuth2UserRealm oAuth2UserRealm(){
        OAuth2UserRealm realm = new OAuth2UserRealm();
        // 不需要加密，直接返回
        return realm;
    }
```

原文链接：Shiro 使用多个 Realm 实现多种登录方式

```
    @Bean
    @DependsOn("lifecycleBeanPostProcessor")
    public UserRealm userRealm() {
        UserRealm userRealm = new UserRealm();
        userRealm.setCredentialsMatcher(credentialsMatcher());
        return userRealm;
    }

    @Bean
    public DefaultWebSessionManager sessionManager(){
        DefaultWebSessionManager manager = new DefaultWebSessionManager();
        manager.setSessionDAO(sessionDAO());
        manager.setGlobalSessionTimeout(10800000);
        manager.setDeleteInvalidSessions(true);
        manager.setSessionValidationSchedulerEnabled(true);
        manager.setSessionValidationInterval(10800000);
        return manager;
    }

    @Bean
    public SessionDAO sessionDAO(){
        return new MemorySessionDAO();
    }

    @Bean(name = "cacheManager")
    public EhCacheManager cacheManager() {
        EhCacheManager cacheManager = new EhCacheManager();
        cacheManager.setCacheManagerConfigFile("classpath:shiro/ehcache-shiro.xml");
        return cacheManager;
    }

    @Bean(name = "credentialsMatcher")
    public CredentialsMatcher credentialsMatcher() {
        return new CredentialsMatcher();
    }

    @Bean(name = "lifecycleBeanPostProcessor")
    public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {
        LifecycleBeanPostProcessor lifecycleBeanPostProcessor = new LifecycleBeanPostProcess
r();
        return lifecycleBeanPostProcessor;
    }

    /**
     * shiro里实现的Advisor类,用来拦截注解的方法 .
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor() {
        AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdviso
();
        advisor.setSecurityManager(securityManager());
        return advisor;
    }
```

```
    @Bean
    @DependsOn({"lifecycleBeanPostProcessor"})
    public DefaultAdvisorAutoProxyCreator advisorAutoProxyCreator() {
        DefaultAdvisorAutoProxyCreator advisorAutoProxyCreator = new DefaultAdvisorAutoPr
xyCreator();
        advisorAutoProxyCreator.setProxyTargetClass(true);
        return advisorAutoProxyCreator;
    }

    @Bean
    public FilterRegistrationBean delegatingFilterProxy(){
        FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();
        DelegatingFilterProxy proxy = new DelegatingFilterProxy();
        proxy.setTargetFilterLifecycle(true);
        proxy.setTargetBeanName("shiroFilter");
        filterRegistrationBean.setFilter(proxy);
        return filterRegistrationBean;
    }

}
```

登录

上面都配置好之后，终于到了登录。

这时候就很简单了，在登录方法中，实例化对应的Token，然后调用subject的login()方法，及时捕异常，没有排除异常代表登录认证成功。

例如传统的账号密码登录，这里的username和password就是前端传的值：

```
        try {
            UsernamePasswordToken token = new UsernamePasswordToken(username, password
;
            SecurityUtils.getSubject().login(token);
            return Result.ok("登录成功");
        } catch (IncorrectCredentialsException ice) {
            return Result.error("密码错误");
        } catch (UnknownAccountException uae) {
            return Result.error("账号不存在");
        } catch (LockedAccountException e) {
            return Result.error("账号被锁定");
        } catch (ExcessiveAttemptsException eae) {
            return Result.error("操作频繁，请稍后再试");
        } catch (ConcurrentAccessException cae) {
            return Result.error("没有权限，无法登陆");
        }
```

而第三方登录就实例化OAuth2UserToken，将第三方平台的类型及其回调的openid等信息保存到tokn中，然后调用subject的login()方法：

```
            // 实例化自定义的授权Token
            OAuth2UserToken.OAuth2User oAuth2User = new OAuth2UserToken.OAuth2User()
```

```
            oAuth2User.setOpenid(authUser.getUuid());
            oAuth2User.setNickname(authUser.getNickname());
            OAuth2UserToken userToken = new OAuth2UserToken(type, oAuth2User);
            // 调用login方法
            SecurityUtils.getSubject().login(userToken);
            servletResponse.sendRedirect("/");
```

结束

将上面代码全部写好之后，启动项目，尝试用户名密码登录和第三方免密登录，如果没有问题则大功成，如果报错了，看官应注意检查是否遗漏代码。

同时注意，上面的代码切忌照搬，还是以理解为主。

如果存在疑问，或者有任何的建议，欢迎评论留言！